Time Parallel Solution of Linear Partial Differential Equations on the Intel Touchstone Delta Supercomputer

Nikzad Toomarian
Amir Fijany
Jacob Barhen

Center for Space Microelectronics Technology
Jet Propulsion laboratory
California Institute of Technology
4800 Oak Grove Dr., MS 303-310
Pasadena, CA 91109

Contact Author:
Dr. Jacob Barhen
Tel: 818-354-9218
Fax: 818-3'33-5013
E-mail: barhen@nips.jpl.nasa.gov

Accepted for publication in

Concurrency: Practice and Experience

February 1994

Time Parallel Solution of Linear Partial Differential Equations on the Intel Touchstone Delta Supercomputer

Nikzad Toomarian, Amir Fijany anti Jacob Barhen

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Abstract

This paper presents the implementation of a new class of massively parallel algorithms for solving certain time-dependent partial differential equations (PDEs) on massively parallel supercomputers. Such PDEs are usually simulated numerically, by discretization in time and space, and by applying a time stepping procedure to data and algorithms potentially parallelized in the spatial domain. In a radical departure from such a strictly sequential temporal paradigm, we have developed a concept of time-parallel algorithms, which allows the marching in time to be fully parallelized. This is achieved by using a set of transformations based on eigenvalue- eigenvector decompositions of the matrices involved in the discrete formalism. Our time-parallel algorithms posses a highly decoupled structure, and can therefore be efficiently implemented on emerging, massively parallel, high-performance supercomputers, with a minimum of communications and synchronization overhead. We have successfully carried out a proof-of-concept demonstration of the basic ideas using a two-dimensional heat equation example implemented on the Intel Touchstone Delta Supercomputer. Our results indicate that linear, and even superlinear speedup can be achieved and maintained for a very large number of processor nodes.

1. introduction

A large variety of physical phenomena can be described by means of Partial Differential Equations (PDEs) [1]. For most practical applications, an analytical solution dots not exist. Hence, numerical solutions of such equations are usually considered. From such a perspective, the development of fast and accurate algorithms has been extensively studied in the literature. Recent advances in massively parallel hardware architectures are highlighting the need for additional advances in this area. Specifically, in order to fully exploit the computing power of these new architectures, existing algorithms must be reexamined based on their efficiency for parallel implementation and, eventually, new algorithms must be developed that, from the onset, take a greater advantage of the massive parallelism.

The Intel Delta, Intel Paragon, and CRAY '1'31) are representatives of an emerging class of massively parallel MIMD architectures. The main feature of this class of parallel architectures is that they provide a large number of very powerful node processors with vector processing capability, but possess a rather simple communication structure (e.g., a toroidal mesh structure for the Delta). More importantly, these architectures allow exploitation of concurrency at two computational levels. That is, in addition to the MIMD parallel computing feature, the vector processing capability of each processor node can be exploited to further increase the overall speedup in the computation. Thus, the design of parallel algorithms for such architectures must result in processes that are coarse grain, can be efficiently vectorized, and require a minimum of communications.

In this paper, we present the implementation of a new class of algorithms for solving a linear parabolic equation (in a bounded domain Ω, with boundary $\partial\Omega$) on a massively parallel supercomputer. Without 10ss of generality, we limit ourselves to a homogeneous, two-dimensional case with Dirichlet boundary conditions. Theoretical extensions to higher dimensions, nonhomogeneous, space dependent coefficients, and different boundary conditions are discussed elsewhere[2].

For the two-dimensional case under consideration, we take the domain to be a square of. length L, i.e., $0 \leq x \leq L$ and $0 \leq y \leq L$. Hence, the parabolic PDE of interest is given as

$$\frac{\partial v}{\partial t} = \alpha\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{I}$$

The pertaining boundary and initial conditions are specified as follows:

$$v(t,x,y) = V(t,x,y) \qquad x,y \in \partial\Omega \qquad 0 < t \leq t_f \tag{2a}$$

$$v(0,x,y) = f(x,y) \qquad x,y \in \Omega \tag{2b}$$

where α is constant and t_f denotes the final time. Superimposing a uniform grid on the domain, i.e., $x = j \times \Delta_x$, $1 \leq j \leq N$ and $y = i$ x Δ_y, $1 \leq i \leq$ AT, and assuming $\Delta_x = \Delta_y = h = L/(N+1)$, will result in discrete values, v_{ij}^k, which approximate the continuous values $v(k\Delta_t, jh, ih)$. In the sequel, the grid points values of v will be referred to either in term s Of the $N \times N$ matrix v_{ij}, or in terms of the N^2 vector v_ℓ, where $\ell = (i-1) \times N + j$, and $1 \leq i,\ j \leq N$.

The discretization of Eq.(1) in both time and space using the above uniform grid yields a family of numerical schemes, formalized as:

$$[I + 2\beta\delta M]v^{[k+1]} = [I - 2(1-\beta)\delta M]v^{[k]} - 2\delta[\beta V^{[k+1]} + (1-\beta)V^{[k]}] \qquad 0 \leq k \leq K \tag{3}$$

In EC].(3), I denotes the N^2 x N^2 identity matrix, $\delta = \Delta_t/2h^2$, where Δ_t is the magnitude of the time step, and $K = t_f/\Delta_t$. The N^2 x N^2 matrix M arises from the discretization of the second order spatial derivatives. By using a five point central differencing scheme, M will be block tridiagonal, given as $M = Tridiag[I, A, I]$, where $A = Tridiag[1, -4, 1] \in \Re^{N}$ 'N. The N^2 vector V incorporates the time dependent boundary conditions, and has the explicit form:

$$\begin{aligned} V = \; & [v_{1,0} + v_{0,1}, v_{0,2}, \ldots, v_{0,N-1}, v_{0,N} + v_{1,N+1}, \\ & v_{i,0}, 0, \cdots, 0, v_{i,N+1}, \qquad\qquad 2 \le i \le N-1 \qquad (4) \\ & v_{N,0} + v_{N+1,1}, v_{N+1,2}, \ldots, v_{N+1,N-1}, v_{N+1,N} + v_{N,N+1}]^T \end{aligned}$$

Finally, the constant β determines the implicit degree of the method. Threw distinct regimes can be considered in terms of β:

1- Explicit method, $\beta = 0$; then Eq. (3) becomes:
$$v^{[k+1]} = (I - 2\delta M)v^{[k]} - 2\delta V^{[k]} \qquad 0 \le k \le K \qquad (5)$$

2- Implicit method, $\beta = 1$; and Eq. (3) becomes:
$$(I + 2\delta M)v^{[k+1]} = v^{[k]} - 2\delta V^{[k+1]} \qquad 0 \le k \le K \qquad (6)$$

3- Crank-Nicholson (C-N) method, $\beta = 1/2$; in this case Eq. (3) can be written as:
$$(I + \delta M)v^{[k+1]} = (I - \delta M)v^{[k]} - \delta(V^{[k+1]} + V^{[k]}) \qquad 0 \le k \le K \qquad (7)$$

We will focus our discussion on the C-N method, with the understanding that the parallel algorithms presented in the sequel arc, in principle, applicable to all three methods. Equations (5-7) represent the marching in time procedure for solving Eel. (1). From a computational point of view the problem is both time and space dependent. Throughout this paper, the: term *space parallel* is used for algorithms that only exploit parallelism in solving Eqs. (5-7) at each time step, while the term *time parallel* refers to algorithms that exploit parallelism in the concurrent computation of all vectors $v^{[k]}$.

The formalism of Eqs. (5-7) appears to imply a strict sequentiality of the computation in time. In recent years, a number of paradigms have been proposed in an attempt to achieve some level of time parallelism. To date, only limited success has been reported[3-8]. In a new development, however, we have presented[9] a concept of time parallel algorithms, which allows the marching in time procedure to be fully parallelized. This is achieved by diagonalizing Eqs. (5-7) through a transformation bad upon the *eigenvalue-eigenvector decomposition* (EED) of the matrices induced by the discretization. Thus, Eqs. (5-7) can be reduced to a set of First Order Linear Recurrences (FOLRs), which allows the solution for al] time steps to be computed concurrently. The resulting time parallel algorithms have a highly decoupled strut.tum and can, therefore, can be efficiently implemented on emerging massively parallel MIMD architectures with minimum communication and synchronization overheads.

This paper is organized as follows. The concept of time parallel algorithms, as applied to the solution of Eq. (1), is described in Section 2. The best known serial algorithm is addressed in Section 3. The specific heat equation, which is used as an illustrative framework for benchmarking the proposed formalism, is presented in Section 4. The results of our numerical simulations on the Intel Touchstone Delta supercomputer are given in Section 5. Finally, some concluding remarks are made in Section 6.

2. Time Parallel Algorithm Description

The time parallel algorithm we propose requires the derivation of the EED of the matrix M. The following theorem (for proof see e.g., [10], p349) is used in the sequel:

Theorem 1. *The EED of an* $N \times N$ *symmetric, tridiagonal Toeplitz matrix* $\psi = Tridiag$ $[b, a, b]$ *is given by*

$$\psi = \theta \lambda \theta. \tag{8}$$

The rows of the matrix θ correspond to the normalized eigenvectors of the matrix ψ, with elements given by:

$$\theta_{ij} = \frac{2}{\sqrt{N+1}} \sin(\frac{\pi ij}{N+1}) \qquad i,j = 1, \cdots, N \tag{9}$$

The $N \times N$ diagonal matrix λ involves the set, of eigenvalues of the matrix ψ, with the values of the i^{th} diagonal element given by:

$$\lambda_i = a + 2b \cos(\frac{i\pi}{N+1}) \tag{10}$$

Here we can note that θ is the one-dimensional Discrete Sine Transform (DST) operator. Hence, it is a symmetric, orthonormal matrix, i.e., $\theta = \theta^T = \theta^{-1}$.

Now, let us define a $N^2 \times N^2$ block diagonal matrix $\Theta = Diag[\theta, \theta, \ldots, \theta]$. Furthermore, we consider the $N^2 \times N^2$ permutation matrix P, which arises in 2-dimensional Discrete Fourier Transforms. The effect of applying P to the N^2 vector with elements v_ℓ is equivalent to transposing the $N \times N$ matrix with elements v_{ij} . Since both matrices Θ and P are symmetric and orthogonal, we have $\Theta = \Theta^T = \Theta^{-1}$ and $P = P^T = P^{-1}$.

Theorem 2. *The matrix M has an EED of the form:*

$$M = \Theta P \Theta \Lambda \Theta P \Theta \tag{11}$$

where, *Λ is a $N^2 \times N^2$ diagonal matrix. The value of each element, i. e., Λ_ℓ, is computed according to:*

$$\Lambda_\ell = -4 + 2\cos(\frac{i\pi}{N+1}) + 2\cos(\frac{j\pi}{N+1}) \tag{12}$$

Proof. From Theorem 1 and the definition of θ, we can see that the matrix M can be expressed as

$$M = \Theta\eta\Theta \tag{13}$$

in which η is a N^2 x N^2 block tridiagonal matrix, given as $\eta = Tridiag[I, \lambda_A, I]$. Here λ_A is itself a diagonal matrix of the eigenvalues of matrix A (see definition of M). Since the block elements of η are diagonal, it can be reduced to a block diagonal matrix as:

$$\eta = PP\eta PP = P(P\eta P)P = P\Psi P \tag{14a}$$

in which Ψ is a $N^2 \times N^2$ block-diagonal matrix, $\Psi = Diag\{\Psi_i\}$. Each $N \times N$ block, Ψ_i, has a symmetric tridiagonal Toeplitz structure given by $\Psi_i = Tridiag[1, \lambda_{A,i}, 1]$. From Theorem 1, the EED of Ψ_i is

$$\Psi_i = \theta\hat{\lambda}_i\theta$$

where the j^{th} element of matrix $\hat{\lambda}_i$ is given by

$$[\hat{\lambda}_i]_j = \lambda_{A,i} + 2cos(\frac{\pi j}{N+1}) = -4 + 2cos(\frac{\pi i}{N+1}) + 2cos(\frac{\pi j}{N+1})$$

If we define $\Lambda = Diag[\hat{\lambda}_i]$, from the definition of Θ it follows that

$$\Psi = \Theta\Lambda\Theta \tag{14b}$$

By substituting Eqs. (14) into Eq. (13) the desired EED of the matrix M, Eq. (11), is obtained.

The definitions of Θ and P imply that the matrix

$$\Phi = \Theta P\Theta \tag{15}$$

is also symmetric and orthonormal, i.e., $\Phi = \Phi^T = \Phi^{-1}$. Note that Φ is the operator of the 2-dimensional sine transform.

our time parallel algorithm is derived by substituting Eqs. (14) and (15) into the C-N scheme of Eq. (7). After some re-arrangement, one obtains:

$$\Phi(I + \delta\Lambda)\Phi v^{[k+1]} = \Phi(I - \delta\Lambda)\Phi v^{[k]} - \delta(V^{[k+1]} + V^{[k]}) \qquad 0 \le k \le K \tag{16}$$

We now define

$$\hat{v} = \Phi v \tag{17a}$$

$$\hat{V} = \Phi V \tag{17b}$$

and, multiplying both sides of Eq. (16) by the nonsingular matrix Φ, we obtain the diagolized form of Eq. (7):

$$(I + \delta\Lambda)\hat{v}^{[k+1]} = (I - \delta\Lambda)\hat{v}^{[k]} - \delta(\hat{V}^{[k+1]} + \hat{V}^{[k]}) \qquad 0 \le k \le K \tag{18}$$

Furthermore, we introduce the N^2 x N^2 diagonal matrix D, with elements

$$D_\ell = \frac{(1 - \delta\Lambda_\ell)}{(1 + \delta\Lambda_\ell)} \qquad 1 \leq \ell \leq N^2 \tag{19a}$$

and define

$$\hat{d}_\ell^{[k+1]} = \frac{\delta(\hat{V}_\ell^{[k+1]} + \hat{V}_\ell^{[k]})}{(1 + \delta\Lambda_\ell)} \qquad 1 \leq \ell \leq N^2 \tag{19b}$$

Recalling the orthonormality of Φ, i.e., $\Phi = \Phi^{-1}$, and substituting Eqs. (17) and (19) into Eq. (1 S), we obtain the recurrence:

$$\hat{v}^{[k+1]} = D\hat{v}^{[k]} - \hat{d}^{[k+1]} \qquad 0 \leq k \leq K \tag{20}$$

which represents a first order inhomogeneous linear system. Note that with $O(K)$ processors, all vectors $\hat{d}^{[k]}$ can be computed in a fully decoupled fashion in a time of O(1). The linear recurrence in Eq. (20) can then be solved in $O(log_2 K)$ by using either a cyclic reduction algorithm[1 1] or a recursive doubling technique[12].

For time independent boundary conditions, one can rewrite Eq. (20) as follows

$$\hat{v}^{[k]} = D^k \hat{v}^{[0]} - [\hat{d}^{[k+1]} + D\hat{d}^{[k]} + \cdots + D^k \hat{d}^{[1]}] \qquad 1 \leq k \leq K \tag{21a}$$

and, since $\hat{d}^{[k]} = \hat{d}$ $\quad (1 \leq k \leq K)$, one obtains the simpler expression:

$$\hat{v}^{[k]} = D^k \hat{v}^{[0]} - \hat{d}\frac{1 - D^k}{1 - D} \tag{21b}$$

Furthermore, if one assumes homogenous b oundary conditions (i.e., $\hat{d} = 0$), the above equation will further reduce to:

$$\hat{v}^{[k]} = D^k \hat{v}^{[0]} \tag{22}$$

It should be emphasized that each processor k can compute its own corresponding power of D in a time of $O(log_2 k)$ without communication (using, e.g., algorithms in [1 3]).

As a simple illustration, we now summarize the time parallel algorithm for constant b oundary conditions. On eacl ı processing node:

o Transform the initial conditions vector, i.e., compute

$$\hat{v}^{[0]} \;,\; \Phi\, v^{[0]}$$

This step can be accomplished in $O(N^2 log_2 N)$ multiply-accumulates, using fast transforms.

o Calculate ea ch vector $v^{[k]}$ using Eq. (22), with a complexity of $O(N^2)$ per time step, k;

o Apply an inverse transform to the! vector z)̂$^{[k]}$, to obtain $v^{[k]}$

$$v^{[k]} = \Phi\hat{v}^{[k]} \qquad 1 \leq k \leq K$$

This step can be accomplished in $O(N^2 log_2 N)$ multiply-accumulates for each time step k, by using fast transforms.

The overall computational complexity of the time-parallel algorithm on a single processor machine is therefore $O(KN^2 log_2 N)$. Because of the inherently decoupled structure of our algorithm, this complexity scales as $0(\frac{KN^2 K log_2 N}{N_p})$ on a system involving N_p processor nodes.

3. Best Serial Algorithm

In order to determine the "best" serial algorithm for the problem under consideration, we make the following observations. The coefficient matrices in Eqs. (5-7) have a symmetric, positive-definite, and sparse structure. This allows the use of rather generic iterative methods such as SO I{, conjugat e gradi ent, etc. [1 4], for solving the linear systems. More importantly, however, we also note that these matrices have similar structures to those arising in the solution of the Poisson equation. In that sense, Eqs. (5-7) represent a sequence of Poi sson equations. Therefore, the so called Fast Poisson Solvers [15] call be directly applied to the linear systems, Eqs. (5-i'), with a greater computational efficiency than the iterative methods [1 G]. In the sequel, we suggest an improved version of the matrix decomposition algorithm of [15].

The computational complexity of such "best serial algorithms" must be evaluated in a framework consistent with the time parallel formalism. We note that such algorithms are also based on the decomposit ion of mat rix M. However, this decomposition is now limited to that specified in Theorem 2. Substituting Eq. (1 1) into the C-N scheme, Eq. (7), and rearranging the terms we obtain:

$$\Theta P(I + \delta\Psi)P\Theta v^{[k+1]} = \Theta P(I - \delta\Psi)P\Theta v^{[k]} \qquad 0 \leq k \leq K \tag{23}$$

Let us define

$$\tilde{v} = P\Theta v \tag{24}$$

Using the orthonormality of Θ and P, and substituting Eel. (24) into Eq. (23), yields

$$(I + \delta\Psi)\tilde{v}^{[k+1]} = (I - \delta\Psi)\tilde{v}^{[k]} \qquad 0 \leq k \leq K \tag{25}$$

One can then rewrite Eq. (25) as follows:

$$(I + \delta\Psi)(\tilde{v}^{[k+1]} + \#1) = 2\tilde{v}^{[k]} \qquad 0 \leq k \leq K \tag{26}$$

Now , if one defines $\tilde{w}^{[k+1]} = \tilde{v}^{[k+1]} + \tilde{v}^{[k]}$, the C-N method call be recast as

$$(\frac{I}{2} + \frac{\delta\Psi}{2})\tilde{w}^{[k+1]} = \tilde{v}^{[k]} \qquad 0 \leq k \leq K \tag{27}$$

$$\tilde{v}^{[k+1]} = \tilde{w}^{[k+1]} - \tilde{v}^{[k]} \qquad 0 \leq k \leq K \tag{2s}$$

Again, let us consider, for the purpose of simple illustration, the case of constant boundary conditions, Then, the best serial algorithm can be summarized in the following three steps:

o Transform the vector of initial conditions, i.e.,

$$\tilde{v}^{[0]} = P\Theta v^{[0]}$$

This can be accomplished at a cost of $O(N^2 log_2 N)$, using fast transforms.

o Calculate the vector $\tilde{w}^{[k]}$ by solving the linear system Eq. (27), and $\tilde{v}^{[k]}$ using Eq. (28); repeat this for all time steps, $k = 1.2', \ldots . K$; the system of $O(N^2)$ linear equations (27) has a symmetric tridiagonal Toeplitz structure: hence, it can be solved in $O(N^2)$ steps;

0 At each time step one needs to output the vector $v^{[k]}$, which is obtained by applying the inverse transformation to $\tilde{v}^{[k]}$:

$$v^{[k]} = \Theta P \tilde{v}^{[k]} \qquad 1 \leq k \leq K$$

This can be accomplished in $O(N^2 log_2 N)$ at each time step k, by using fast transforms.

Thus, the overall computational complexity of the best serial algorithm cm a single processor is $O(KN^2 log_2 N)$.

4. The Heat Equation

In **order to** provide a concrete framework for assessing the potential of our proposed approach to time parallelism, we focus our attention on a two-dimensional heat conduction problem modeled by a linear parabolic PDE[1 7]. This problem has the advantage of exhibiting both sufficient computational complexity, and possessing analytical solutions. Furthermore, it has been widely used for benchmarking of parallel algorithms[4-8,1 8-20].

'I'o fix the ideas, consider the case of transient conduction in a long bar having a square cross section, of thickness L. The bar is assumed to be infinite in the z direction, so that the heat profile will wary only in the x and y directions. For simplicity, we furthermore assume that the cross sectional temperature, $v(t, x, y)$ is given at time $t = 0$ by

$$v(t, x, y) = sin(x/L) \cdot sin(y/L) \tag{29}$$

where $0 \leq x \leq L$, $0 \leq y \leq L$. The temperature of the bar at the boundaries is kept constant, i.e.,

$$v(t,x,0) = v(t,x,L) = v(t,0,y) = v(t,L,y) = 0 \tag{30}$$

Thus, the differential equation to be solved is:

$$\frac{\partial v}{\partial t} = \alpha\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{31}$$

where the constant α is the thermal diffusivity. The initial temperature distribution is a product of two functions, each of which involves only one of the independent space variables. Hence, using separation of variables[17], the temperature distribution in a cross section of the bar can be found to be:

$$v(t,x,y) = e^{-2(\pi/L)^2\alpha t}\, sin(\pi x/L) \cdot sin(\pi y/L) \tag{32}$$

This analytical expression will be used to validate the numerical results of the implementation of our time parallel algorithm on the Intel Touchstone Delta supercomputer.

5.1 Implementation Results

In order to evaluate the potential of our proposed time parallelism paradigm, a FORTRAN computer code for solving the 2-D heat equation [i.e., Eqs. (29-31)] was written and implemented on the Intel Touchstone Delta. The numeric nodes of the Delta are i860 microprocessors operating at 40 MHz. These nodes are rated at 80(peak) single precision MFLOPS.

For the actual simulations, we selected the Crank-Nicholson scheme, i.e. we set $\beta = 0.5$, and assumed a thermal difusivity of 0.1. The bar thickness, L, was partitioned, in each spatial direction, into $N + 1$ segments using $N + 2$ equidistant grid points. Since the boundary points have a fixed value in this problem, there are only N^2 lattice points at which a computation is performed. The spatial grid size and the time step size were chosen as $h = \Delta_x = \Delta_y = 0.1$ and $\Delta_t = 10^{-4}$ respectively.

In order to enable a more accurate measurement of the computation time at each processor, and to allow for potential inaccuracies stemming from the numerical scheme to accumulate, we report results after $K = 5000$ time steps, i.e., 0.5 second after experiment start up. This number of time steps was divided between the N_p processors in the following manner. The p^{th} processor calculates the k^{th} time step, where $k = p + m \times N_p$, $0 \leq m \leq K/N_p$, and $k \leq K$.

On the Delta machine, processors are allocated in terms of a rectangular mesh. In our implementation, a square partition was generally used, in which the number of columns and rows was simultaneously varied from 1 to 10. For the case of 120 processors, a 10 x 12 rectangular mesh was employed. Each node program starts by determining its node ID and the number of processors in the partition. Following the identification procedure, the first node (node 0) reads, from the screen, the lattice size, N, of the problem to be solved, and broadcasts it to all other nodes. At each node, an identical filename is constructed, "based upon the lattice size, N, and the number of nodes, N_p. Then a file with this specific filename is opened, which is shared between all nodes, and used for recording the initialization and computation time of each node. On the Delta machine, four different modes are available for accessing a file. We have used the third mode, which requires a fixed record size. Each node has its own file pointer and all READ and WRITE operations are ordered by node number. Here, setting the mode and closing the file are synchronous operations. 'l'bus, we start timing immediately after setting the mode.

To proceed, the values of the parameters α, β, Δ_t, h and K are initialized, the initial temperature distribution] and its transformation are calculated, and the values of D_ℓ^p and B_ℓ, where $B_\ell = D_\ell^{N_p}$ $(1 \le \ell \le N^2$ and $0 \le p \le N_p - 1)$ are computed. At this stage we clock the time again and subtract it from the starting time to obtain the initialization time. This time is a constant across all nodes, function of the spatial resolution, N. It is measured to be on the average 12, 48, 190, 756 milliseconds per node, for N equal to 15, 3.1, 63, 127 respectively.

Following initialization, a DO loop is executed, 011 each processor. Each loop index runs from p to K, in increments of N_p. First, each processor p calculates the temperature distribution of the p^{th} time step using Eq. (22) and the D values currently in memory. Then, the value of D at each node is updated according to the formula

$$D_\ell = D_\ell \times B_\ell \qquad 1 \le \ell \le N^2$$

yielding the quantities required for computing the temperature at the $(p + N_p)^{th}$ time step. After processing $O(K/N_p)$ time steps, the measured times for initialization and computation are written from each node onto the common file.

'['able 1 shows the total (i.e., initialization plus computation) time, in milliseconds, for four different cases involving different lattice grid sizes. The first row of this table displays the time achieved with the best serial algorithm (see Section 3) using a single node of the Delta machine. The other mws present the average time per node, calculated according to

$$\tau_{ave} = \frac{1}{N_p} \sum_{p=0}^{p=N_p-1} \tau_p$$

where τ_p is the total' time posted for processor p. The speedup (i. e., best serial algorithm processing time divided by τ_{ave}) achieved as a function of the number of nodes is displayed in Fig. 1.

As can be seen from Fig. 1, even for a small grid size of $N = 15$, the time-parallel algorithm achieves about two orders of magnitude speedup by using 120 processors. Note that, for small N, there is only a limited spatial parallelism in the computation. This is consistent with reported results of the space-parallel solution of this model problem in [18]. Interestingly, for larger A', (e.g., $N = 127$,) the time-parallel algorithm achieves a **superlinear speed up**, i.e., a speed up greater than the number of processors. To understand this behavior, it is important to recall the fact that the time-parallel algorithm also exploits a second level of concurrency, by taking advantage Of the vector processing capability of the nodes of the Delta. Although both algorithms have been implemented in a straightforward fashion, i.e., by using the automatic vectorization capabilities of the Intel Delta, for large N the basic operations of the time-parallel algorithm become more suitable for vector processing, and thus are executed faster by the *i*860 nodes. In fact, for large N, the time-parallel algorithm is computed faster on a single *i*860 processor than the best serial algorithm, even though it requires a greater number of operations.

A comparison of our results with both theoretical and practical results reported in [4-8,18-20] for the same model problem clearly highlights the efficiency of our time-parallel computing approach. Our results also demonstrate that, even with a limited number of processors, it is more efficient to exploit parallelism in time than in space.

5. Conclusions

In this paper, we have presented a novel time parallel algorithm for the solution of linear parabolic partial differential equations. The basic idea is to use a transformation involving the eigenvalue-eigenvector decomposition of the coefficient matrices induced by the discretization process. The resulting diagonalization yields a decoupling of the time stepping scheme, which in turn allows the solution for all the time steps to be computed in parallel.

At first glance, it might have seemed that time-parallel algorithms could be more efficiently applied to those problems for which the analytical expressions of eigenpairs of the coefficient matrices are known, and hence no computation is needed. However, our preliminary analysis of the performance of time-parallel algorithms for the solution of the Schrodinger equation, for which additional computations are required for derivation of such eigenpairs, appears to clearly indicate the contrary[2]. The result seems rather general and shows that, for most cases, *the performance of the time-parallel algorithms we propose, will not be reduced due to the need of computing the eigenpairs, if the latter is performed efficiently.*

Acknowledgments The research described in this paper was performed at the Center for Space Microelectronics Technology, Jet Propulsion Laboratory, California Institute of Technology. It was jointly sponsored by Innovative Science and Technology Office of the Ballistic Missile Defense Organization, and by the National Aeronautics and Space Administration, Office of Advanced Concepts and Technology. The support and encouragement of Dr. Paul Messina, Director of the Concurrent Supercomputing Consortium, is greatly appreciated.

References

1. Whithman G.B., *Linear and Nonlinear Waves*, John Wiley and Sons, New York, (1974).

2. Fijany, A., J. Barhen and N. Toomarian, "Fast Time and Space Parallel Solution of the Schrodinger Equation", submitted to *SIAM J. Scient. Comp.*, February 1994.

3. Lelarasmee E., A. Ruheli, and A. L. Sangiovanni-Vincentelli, "Tile Waveform Relaxation Method for the Time Domain Analysis of Large Scale Integrated Circuits", *IEEE Trans. Computer-Aided Design*, Vol. 1, pp. 131-145, 1982.

4. Saltz J. H. and V. K. Nail, "Towards Developing Robust Algorithms for Solving Partial Differential Equations on MIMD Machines", *Parallel Computing*, Vol. 6, pp. 19-44, 1988.

5. Womble D. E., "A Time-Stepping Algorithm for Parallel Computers", *SIAM J. Sci. Stat. Comput.*, Vol. 11, No. (5), pp. 824-837, IWO.

6. Hackbusch W., "Parabolic Multigrid Methods", Procs. 6th Int. Symp. on Computing Methods in Applied Sciences and Engineering, December 1983.

7. Horton G. and R. Knirsch, "A Time-Parallel Multigrid-Extrapolation Method for Parabolic Partial Differential Equations", *Parallel Computing*, Vol. 18, pp. 21-29, 1992.

8. Strang G. and G. J. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

9. Fijany A., "Time Parallel Algorithms fen' Solution of Linear Parabolic PDEs", Procs., 1993 International Conference on Parallel Processing, Vol. 3, pp. 51-55, August 1993.

10. Barnett S., *Matrices: Methods and Applications*, Clarendon Press, 1990.

11. Hockney R. and C. Jesshope, *Parallel Computers*. Adam Hilger Ltd., 1981.

12. Kogge P. M. and 11. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations", *IEEE Trans. Comp.*, Vol. C-22(8), pp. 786-793, 1973.

13. Knuth, D. E. *The Art of Computer Programing, Vol. 1: Fundamental Algorithms* Addison-Wesley, Reading, MA, 1968.

14. Varga R. S., *Matrix Iterative Analysis*, Prentice-Hall, NJ, 1962.

15. Buzbee B., G. Golub, and C. Nielson, "On Direct Methods for Solving Poisson Equations", *SIAM J. Numer. Anal.*, Vol. '7, PP. 627-656, 1970.

16. Swarztrauber P. N. and R. A. Sweet, "Efficient Subroutines for the Solution of General Elliptic and Parabolic Partial Differential Equations", *A tmospheric Technology*, pp. 79-81, September 1973.

17. Chapman, A . J., *Heat Transfer*, Macmillan, New york, 1967.

18. Gallopoulos E. and Y. Saad, "On the Parallel Solution of Parabolic Equations", Proc. ACM Int. Conf. on Supercomputing, pp. 17-28, June 1989.

19. Rodriguez G. and D. Wolit zer, "Preconditioned Time-Differencing for the Parallel Solution of the Heat Equation", Proc. 4 t h SIAM Conf. on Parallel Processing, pp. 268-272, 1 990.

20. Vandewalle S. and R. Piessens, "Efficient Parallel Algorithms for Solving Initial-Boundary Value and Time-Periodic Parabolic Differential Equations", *SIAM J. Sci. Stat. Comput.* , Vol. **13(6),** pp. 1330-1346, November 1992.

Figure Caption

Fig. 1: Speedup of the time-parallel algorithm as function of the number of processors for different mesh size.

Table 1: Total execution time, in milliseconds, for different mesh size and number of])100?ss01s employed.

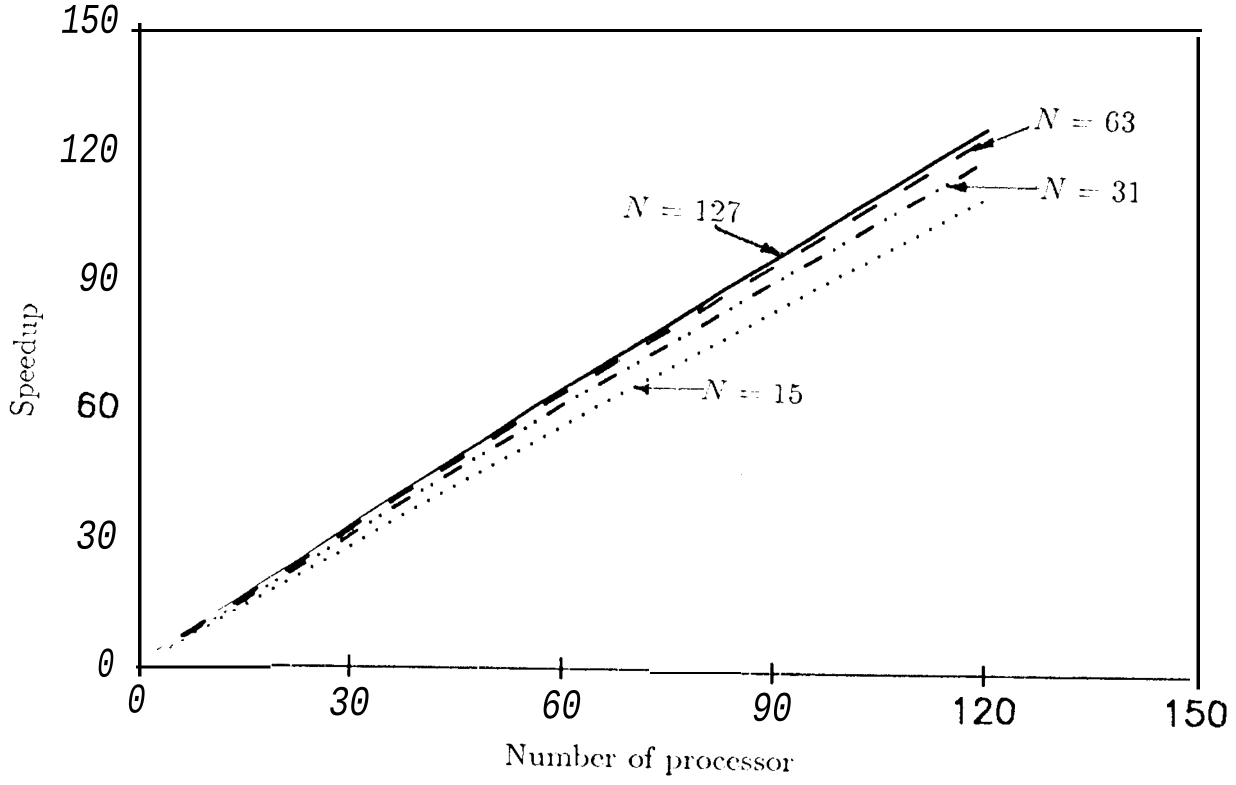

Fig. 1: Speedup of the time-parallel algorithm as function of the number of processors for different mesh size.

Table 1: Total execution time, in milliseconds, for different mesh sizes and number of **processors employed**

Number of	Dimension of the Mesh			
Processors	15	31	63	127
1	30364	119453	473665	1898395
1	31208	118109	429817	1693977
4	7811	28642	107595	424074
9	3478	12620	47925	188885
16	1962	7119	27040	106576
25	1260	4574	17373	68475
36	879	3191	12123	47784
49	649	2357	8957	35307
64	500	1816	6903	27210
81	389	1447	5500	21680
100	324	1181	4491	17703
120	272	992	3774	14877